

# Build Your Own Device

**Angelo Dureghello**  
Sysam Firmware Solutions

Embedded Linux Conference & OpenIoT Summit Europe 2016 - October 11-13th 2016, Berlin, Germany

**Sysam Firmware Solutions**  
[www.sysam.it](http://www.sysam.it)

Linux kernel, device drivers, embedded system design,  
sw development and consulting, hardware prototyping  
and board wake-up support



Speaker

## Angelo Dureghello

Embedded Systems Engineer, Sysam - Nomovok Oy

- from Trieste, Italy, with high interest for electronic, hw repair and computer programming from high school age,
- graduate in Telecommunications, experienced low level programming, C, C++, worked in several different architectures, experience in Linux kernel and drivers debug, driver development, hw repair and troubleshooting, some experience in electronic design,
- actually working for Nomovok Oy, Finland
- U-Boot m68k/Coldfire architecture custodian,
- some contribution to Linux kernel for the Coldfire architecture.
- member of “mittelab” hackerspace and LUG in my hometown, Trieste

*Aim of this talk is to present an alternative way to explore the interesting, funny and lucrative embedded world:  
Build Your Own (Linux) Device.  
So the aim is to offer some basic guidelines for this challenging and exciting task.*

Of course, there can be complex steps to pass, studies and errors to take in account and some money to spend.

A kind of little investment but there can be personal motivations.

*Ok, but what could be the benefits, at the end ? I buy a module for few dollars, certainly more advanced of a board done at home, and no time is spent.*

- to grow an additional skill on the hardware level, designing, troubleshooting and understanding the circuit interconnections, how peripherals protocols works, and to enjoy bringing components up to work properly, one by one,
- to select the combination of CPU / SoC and additional components where you like to increase your experience,
- maybe a way to practice and contribute in Linux development, writing / adapting drivers, or fixing any issue found during the wake-up of the components,
- of course, customizing

Example open board: “amcore”, 100Mhz mcf5307 CPU (no mmu), 4MB parallel NOR flash and 16MB SDRAM



Maybe not few, they could be:

- some basic electronic knowledge, like to know the difference between AC and DC, the Ohm law, what are resistors, capacitors, transistors, diodes, how resistors / capacitors in series and parallel behave, what are pull-up and pull-down resistors, and the like, things that can be learned on the way, with some freenode irc #electronics help, plenty of tutorials, books, or asking clarifications to some expert friend,
- some minimal measurement equipment, as a multimeter and an oscilloscope,
- a solder station with a conic and quite thin tip, possibly with hot air too,
- solder paste, tin, some alcohol, a lens, tweezers, manual pump for tin removal, and other small accessories to buy on the way,
- some manual soldering practice, can be done de-soldering and re-soldering components on old broken boards as routers, etc..

# Requirements



**A whole laboratory is not necessary initially, it can be improved gradually in years.**

**Finding an old table where you can practice soldering, initially, should be enough.**

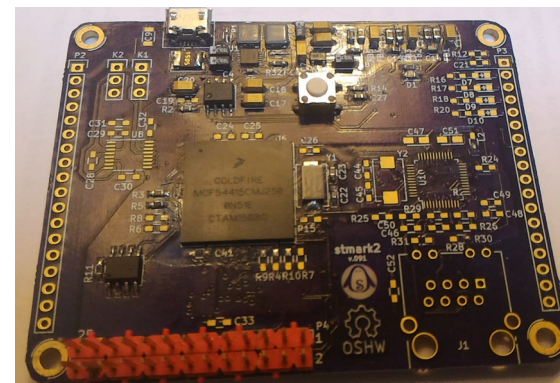
**In case a small investment want be done, a good candidate is the oscilloscope, it is likely the most used and useful debug instrument.**

**A good one can really help a lot and speed up the Things. Look for at least 1Mpoints of memory.**



# Step 1 - CPU selection

- you can select a Linux - supported CPU where you like to grow your experience, of course with some limitations,
- QFP package - easier to solder, for BGA, it can still be done but at greater difficulty,
- clock frequency, the higher clock you want to go, the harder will be the things. Complications comes moving to higher speed of the bus clock frequency. A good start can be to stay between 50 and 100Mhz for the bus clock,
- a look on the price, power supplies, programming interface, if parts are available (samples ?), development boards, application notes, documentation, etc,



## Step 2 – non volatile memory

Selecting a non-volatile memory to boot. The first boot binary (generally a boot loader) is loaded into RAM from the internal CPU ROM loader, some time can be executed (XIP) too.

- you need to check what boot types are supported from the CPU / SoC. Modern SoC's have several options, parallel NOR Flash, SPI Flash, NAND, USB, SD etc etc, while simpler / older CPU's as mcf5307 offers a single choice.
- SPI NOR are generally in packages easy to solder,
- NAND Flash chips are also available in TSOP packages, but introduce the ECC issues
- a good way to start could be a parallel NOR,
  - packages 0,5mm pitch easy to solder,
  - parallel bus pinout easy to understand,
  - CPU can execute in place (XIP) so external SDRAM init can happen just fetching instruction by instruction directly from the FLASH until the point, after SDRAM init, relocation to SDRAM can eventually happen.

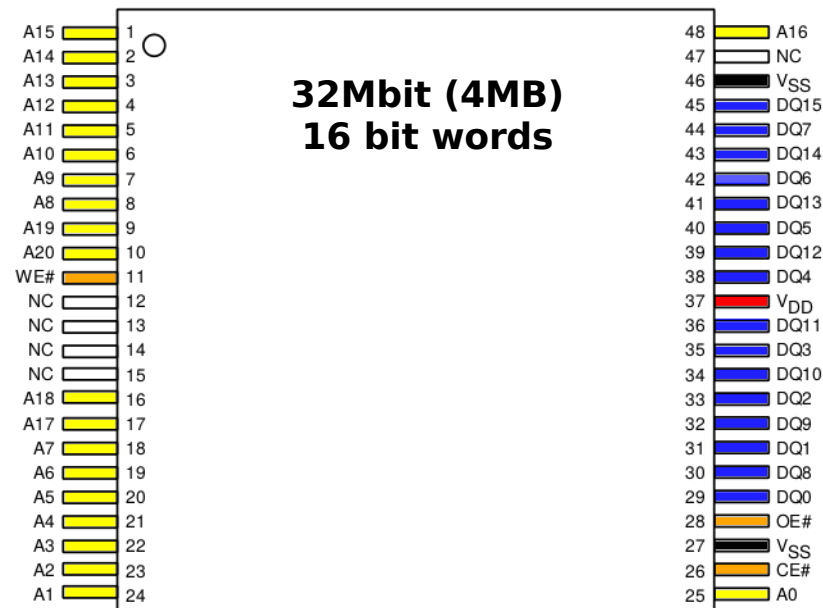
# Step 2 – non volatile memory

## Parallel NOR Flash

- needs a special sequence for erase and write operations,
- not that fast, but it's simple to read,

```
volatile f_uint16 *ptr = 0x100;
f_uint16 read_value = *ptr;
```

CPU Operation	Address bus	Effect
Set address	0x100	nothing
CE# Low	0x100	Chip selected
OE# Low	0x100	Data on bus after time "t"
CE# OE# High	---	Chip disabled
About "t", check CPU wait states		



- this "amcore" board selected model is a 16 bit word addressable Flash

## Step 2 – non volatile memory

- as non-BGA package, you have NOR parallel flash until 1 Gb (256MB)
- parallel NOR are erased in blocks, but can be randomly read,
- parallel NOR throughput is known to be not high, with some tricks can be improved a bit,

# Step 3 – external RAM

Depending on the CPU, it can be SDRAM, DDR, DDR2, 3 etc. For this sample board, mcf5307 forces to use a SDRAM, so a 16MBytes SDRAM has been selected

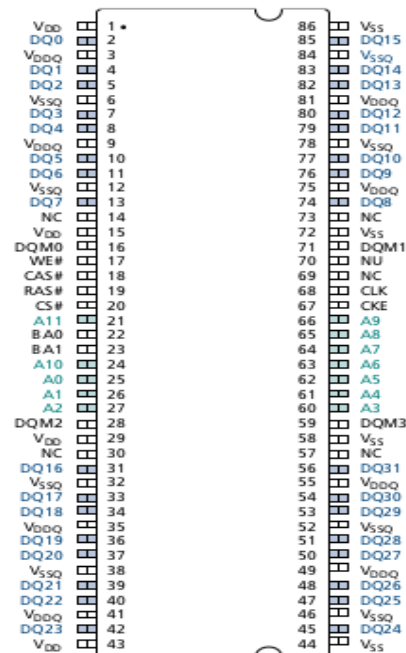
- dynamic RAM are formed from transistor/capacitor cells, organized in columns and rows, and require refresh cycles from CPU (while SRAM not),
- refresh is auto-magically handled from CPU,
- understanding how column / row addressing works can be not that simple,
- fortunately, CPU datasheets reports tables that says, depending on the SDRAM model to connect, how the CPU-SDRAM connections should be done.

Table 2: Configurations

	4 Meg x 32
Configuration	1 Meg x 32 x 4 banks
Refresh count	4K
Row addressing	4K (A0-A11)
Bank addressing	4 (BA0, BA1)
Column addressing	256 (A0-A7)

Table 11-26. MCF5307 to SDRAM Interface (32-Bit Port, 8-Column Address Lines)

MCF5307 Pins	A15	A14	A13	A12	A11	A10	A9	A17	A18	A19	A20	A21	A22	A23	A24	A25	A26	A27	A28	A29	A30	A31
Row	15	14	13	12	11	10	9	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Column	2	3	4	5	6	7	8	16														
SDRAM Pins	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16	A17	A18	A19	A20	A21

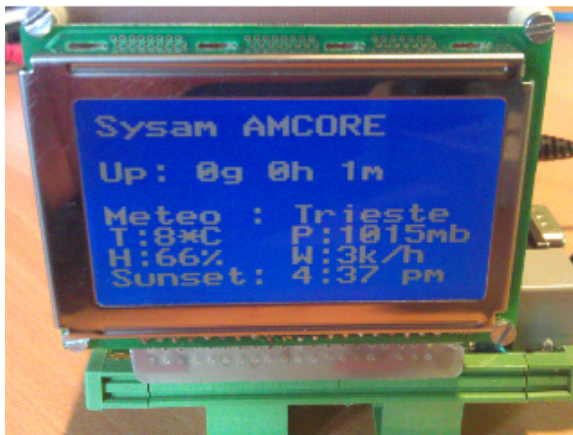


# Step 4 – customizations

In this step you decide how to customize the board, first main components (CPU, FLASH, RAM) has been selected, so now comes the "specialization" of the board.

I.e., the "amcore" reference board for this sessions still adds:

- a parallel network MAC + PHY chip,
- a i2c RTC chip + cell battery,
- a generic connector to expand functionalities of the board in a later step,



Simple meteo monitor



128x64 Graphic LCD expansion

- other peripherals are generally added in embedded applications, as SD CARDS, i2c serial eeproms, audio codecs, accelerators, gps, wifi modules, USB, etc etc etc, but they of course must be supported from the CPU.

# A development approach

**1) Choose a CAD**

**2) Schematic diagram**

**3) Electric Error check and module assignment**

**4) move to PCB CAD side**

**5) route all the connections**

**6) perform PCB error checks**

**Ok, proceed for the “manufacturing”**

**7) generate “gerber” files**

**8) look for best price/quality PCB prototype producer**

**9) order, and after some days you will receive your PCB**

**10) solder components, debug for errors step by step.**

**In case of hw issues, you can fix them with temporary hacks, and then, go back to point 7.**

# 1 - Choose a CAD

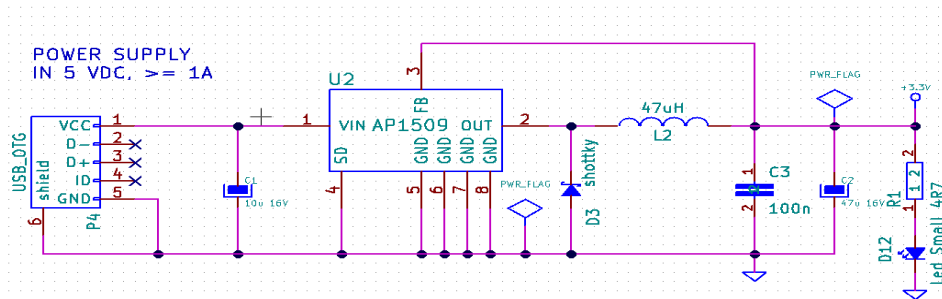
**Cheap CAD or free / opensource:**

- **Kicad (win, mac, linux), opensource, free, irc/freenode #kicad**
- **non-free, closed, but quite cheap license,**
- **several others**
  
- **other non-free CAD's are available for every prices**
  
- **CAD is just a tool.**

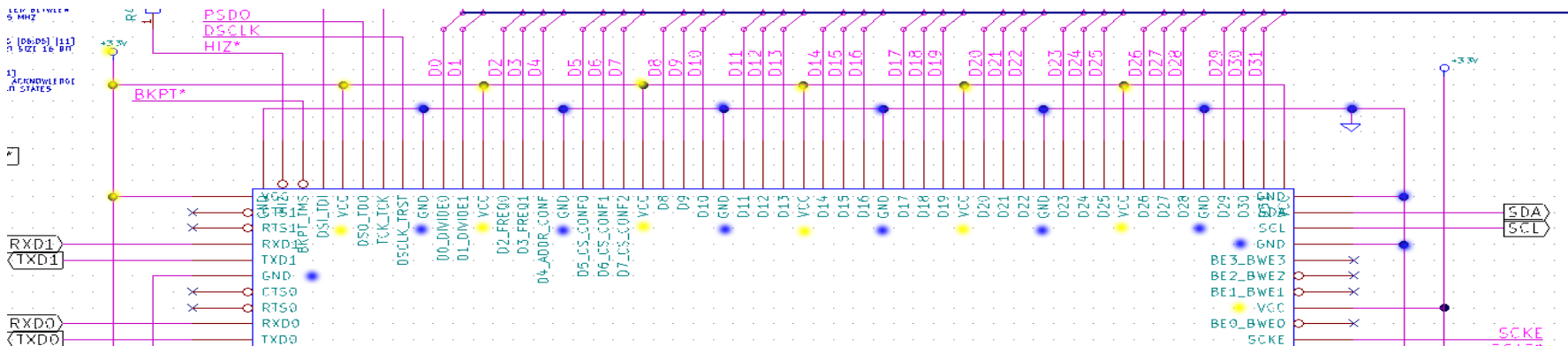
**For the “amcore” board, Kicad has been selected.**

## 2 - Schematic diagram

A way to start is from power supply, from where all the current is coming, and then connecting power to CPU. So now a day you would go with a buck converter:

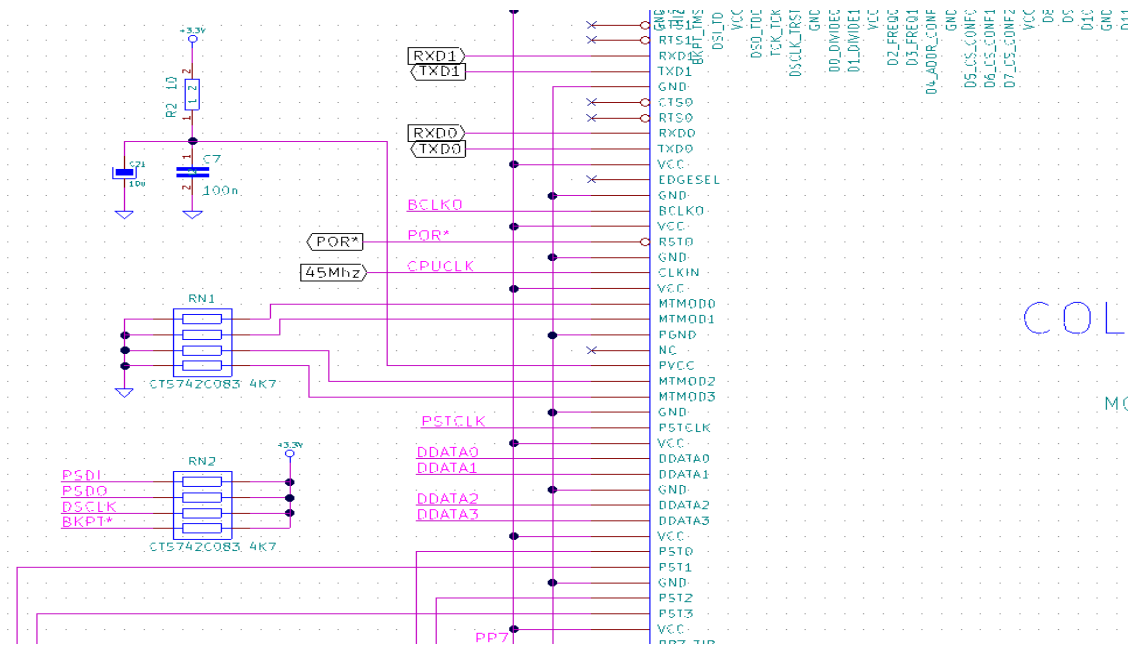


- good point, mcf5307 just needs 3.3V,
- check input-output drop,
- filtering capacitors, see ESR ordering them,
- bypass (typ 100n) capacitors,
- power led



## Connect oscillator, pull-up/down, bootstrap options, debugger interface

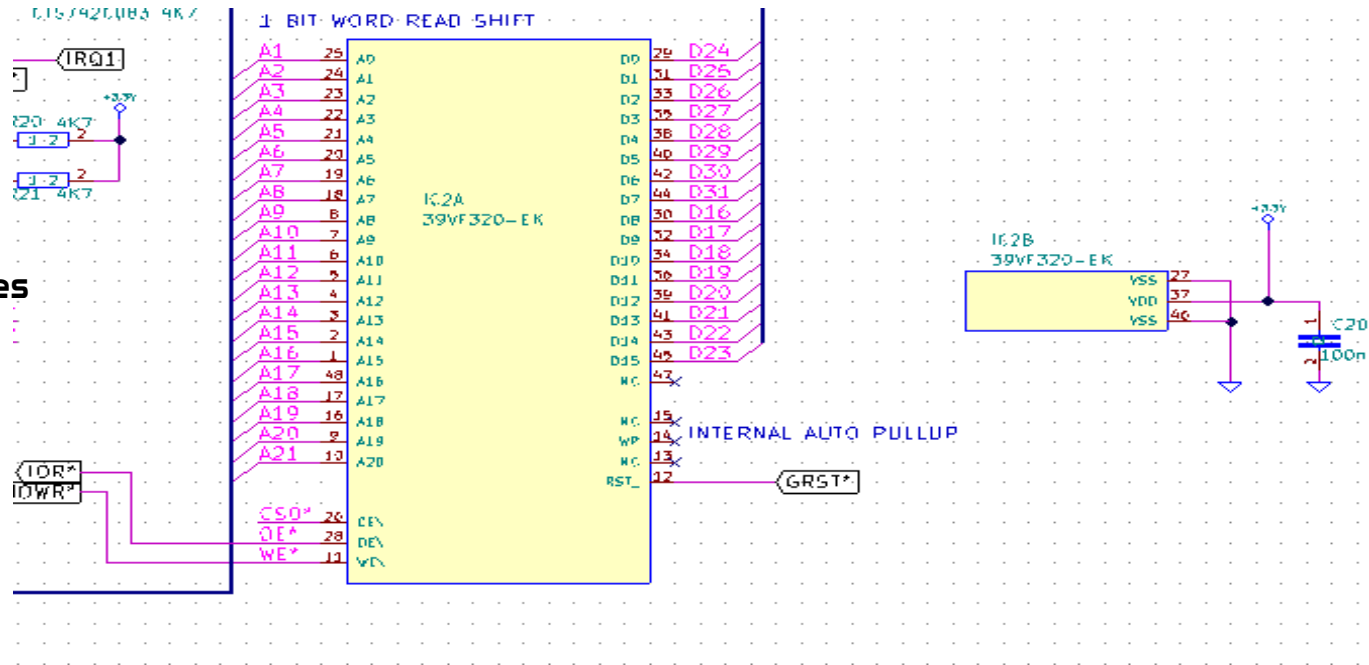
- **checking CPU datasheet to know what are all the CPU connections to be done can be a long task, that can introduce errors.**
- **diagrams of evaluation or similar boards comes into help (copying is often ok, understanding the reason of each copied part improves the knowledge).**



## 2 - Schematic diagram

- connect the non-volatile program memory
- using a parallel NOR, just wire the address bus, data bus, CE#, OE#, WE#, VCC, GND
- note that in this application, CPU A1 is connected to A0 of the Flash (16 bit flash). Also, CPU is BE.

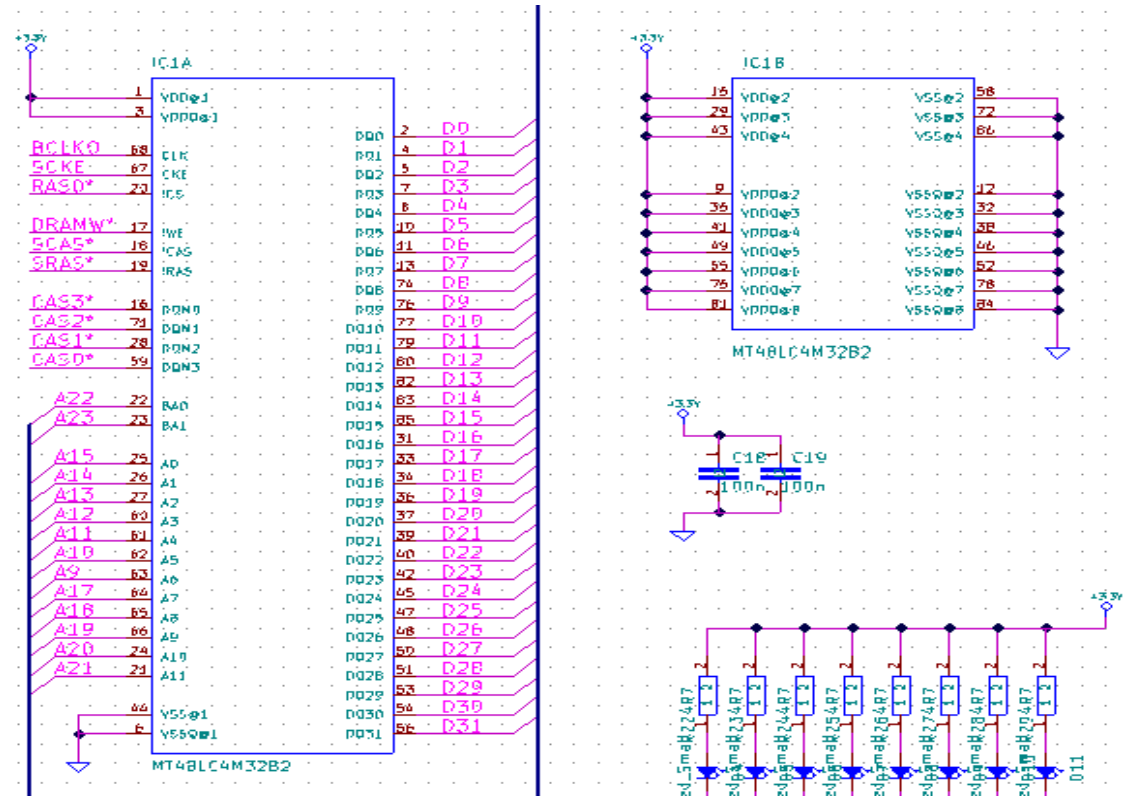
$$2^{22} = 4\text{MBytes}$$



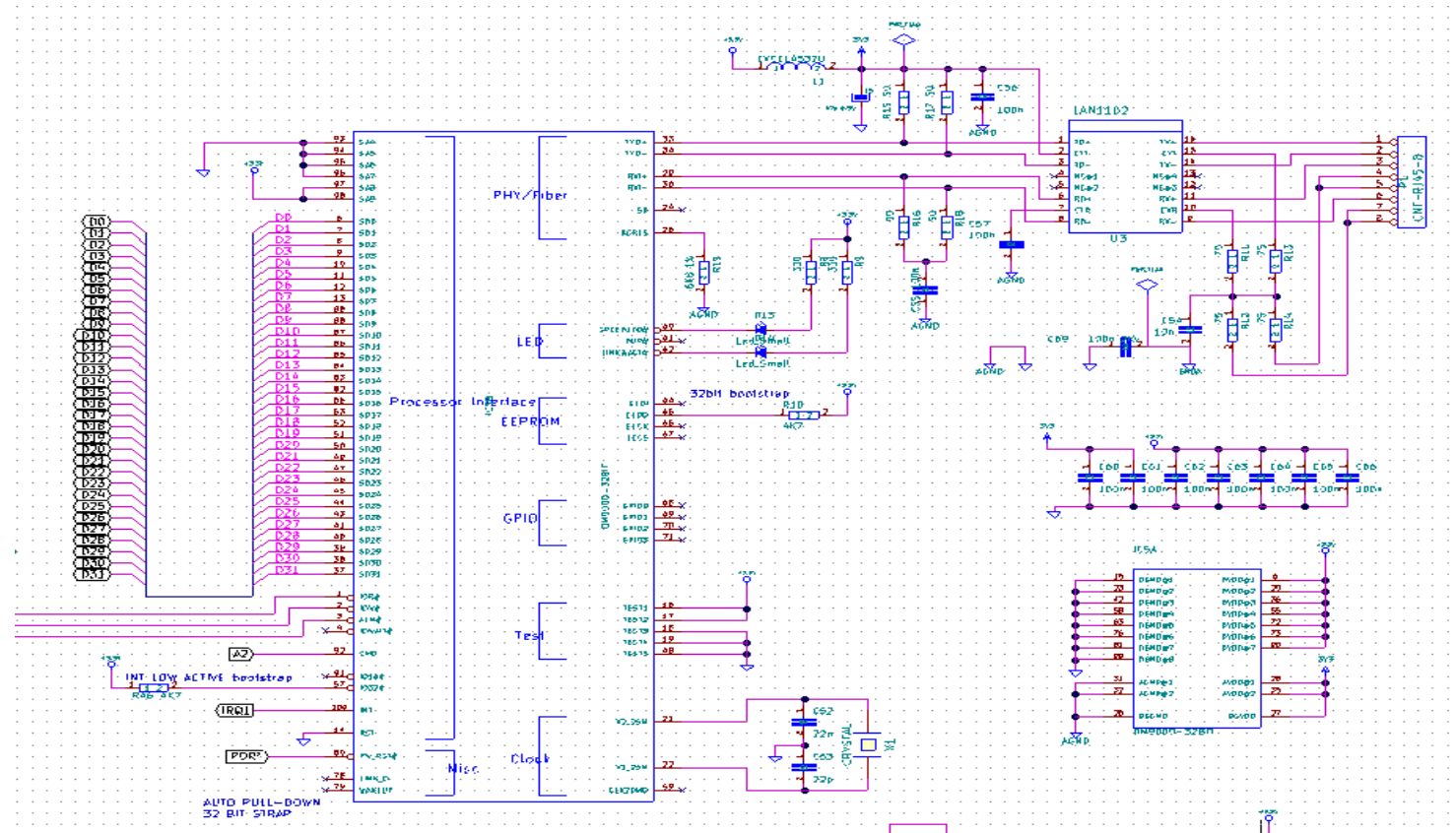
## 2 - Schematic diagram

### Connecting the SDRAM memory

- connecting SDRAM can be difficult, even looking on tables supplied from the datasheet,
- development boards schematics are a good basic guideline, double check that all is correct,
- different configurations from development boards as using multiple banks, requires some more experience, and will introduce also SW configuration issues,



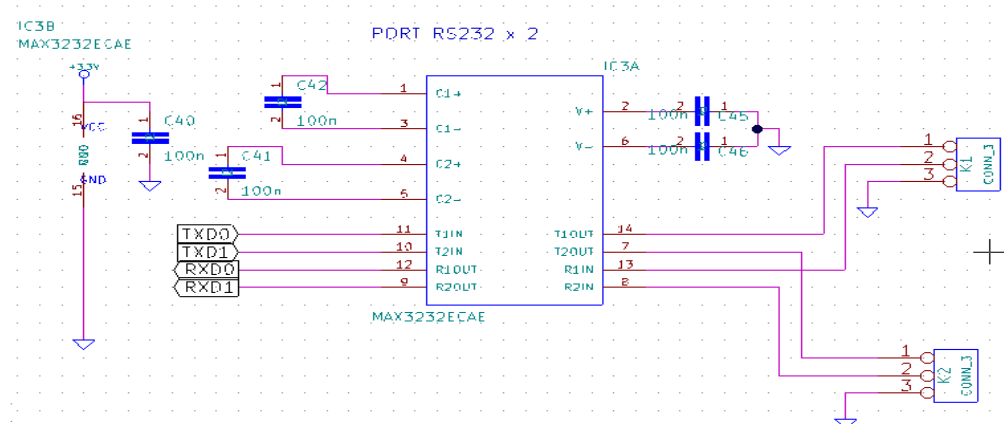
- **Select your special “accessories”,**
- **follow datasheets end especially applications notes,**
- **customization can still be done in a further step, if a generic expansion slot is added**



## 2 - Schematic diagram

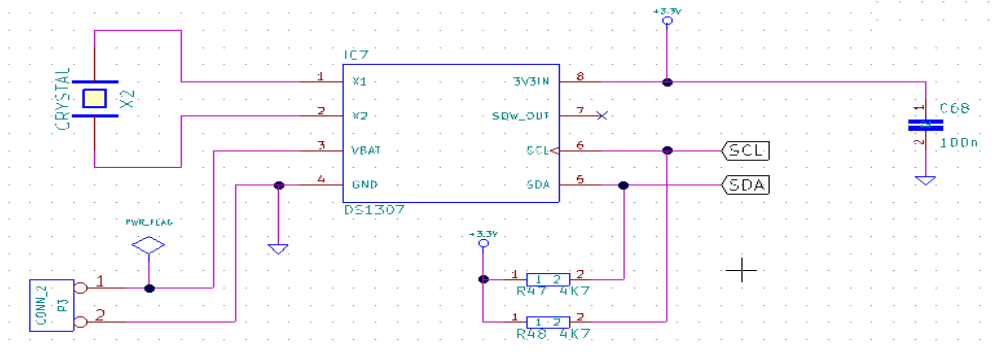
### CUSTOMIZING

- one UART console port can be added for debug purposes
- i.e. on “amcore” board, an i2c RTC has been also added
- check application notes of additional IC's



### Finally

- just complete the electrical error check (ERC),
- perform other needed passes as assigning pcb modules and move to PCB.



## PCB DESIGN

- to start, it is not mandatory to be a professional and very experienced hardware designer,
- with some attentions and avoiding too high bus frequencies, a basic board even without a perfect design should work.
- about how to design a PCB, there several good tutorials around.
- before you start you should focus on a PCB manufacturer, there is a big number of producers all over the world, each of them have some constraints (min vias / track / drill sizes) i.e., you can go to some instant-quote sites, insert your board data and check
- here the “amcore” used parameters

	Clearance	Track Width	Via Dia	Via Drill	uVia Dia	uVia Drill
Default	0,123	0,122	0,5	0,2	0,2	0,1

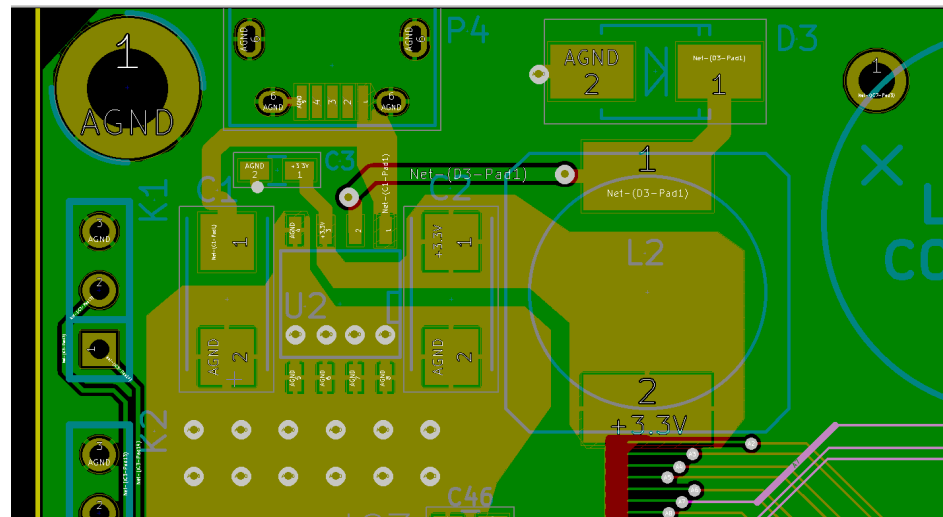
## PCB DESIGN : JUST SOME HINTS

Entering the hardware design techniques is too wide for this talk session, and there are plenty of good tutorials and application notes.

- a big ground plane could be on the whole 4<sup>th</sup> layer, so each ground connection can be easily done by a vias from the top layer, it also helps on electromagnetic shielding, helps for power dissipation, and the like,
- power supply can stay in the top layer,
- all other digital signals could possibly stay in 2<sup>nd</sup> and 3<sup>rd</sup> middle layers,
- the above is not the more proper way but works. Some memory constructors suggest 6 layers and to keep each signal layer next to a ground layer, etc etc. Increasing knowledge on PCB design will help anyway.
- resistors, 100n bypass capacitors, is possible to use hand-soldering pads
- 100nF BYPASS CAPACITORS: bypass capacitors must always be physically located near the IC to protect
- oscillator, follow, if any, the datasheet directions

## PCB DESIGN : BUCK CONVERTER

- using a micro / USB connector can be convenient, (a fuse can be enough, no inversion protection needed)
- arrange a good grounding to help heat dissipation,
- set correct width for tracks that need keep high currents,
- there is much more hardware design theory to improve knowledge as you prefer.



## PCB DESIGN : SDRAM ROUTING, TERMINATION RESISTORS ?

- there can be the need of terminators for both long tracks (reflection) or on short lines to avoid “rings”. There is a huge high-speed signals theory behind, source series, middle or end parallel terminations, but let’s try to keep it simple,
- some readings on memory constructor application notes can be useful, you can also see about this the famous “High Speed Digital Design” - a Handbook of Black Magic – (Johnson – Graham),
- checking development boards here is very important. At least that solution works,
- considering only the short line rings issue, a series resistor near CPU is generally fine. You can set up PCB for it,
- on DDR memories things are a bit different, there are VTT and Vref involved and similar termination strategies are used, but signal speed generally grow. More study on PCB design rules is required.

## PCB DESIGN : SDRAM ROUTING, SOURCE TERMINATION RESISTORS ?

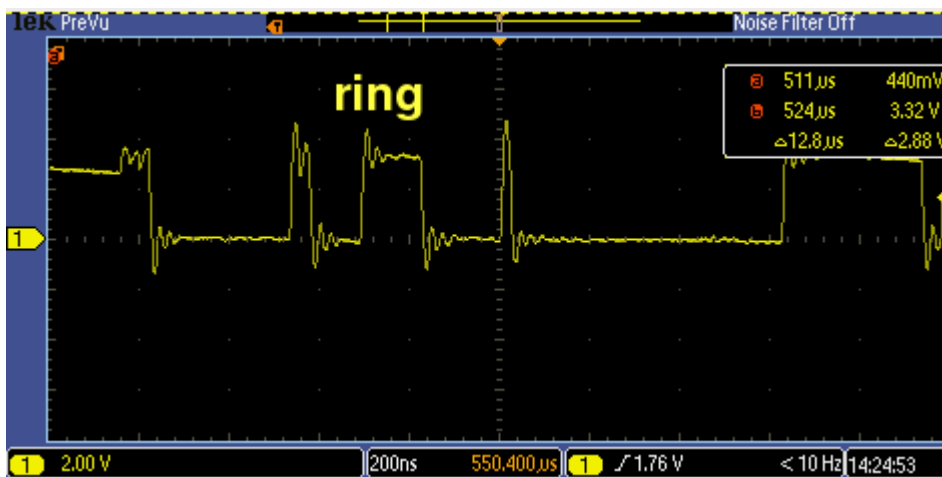
- things can work even without any resistor,
- some basic rules:

$$\text{CPU } Z_o + Z_{\text{term.}} = Z_{\text{line}}$$

For  $Z_o$ , you have to find a compromise:

$$Z_{oh} = (V_{DD} - V_{oh}) / I_{oh}, Z_{ol} = V_{ol} / I_{ol}$$

For  $Z_{\text{line}}$ , there are calculators.

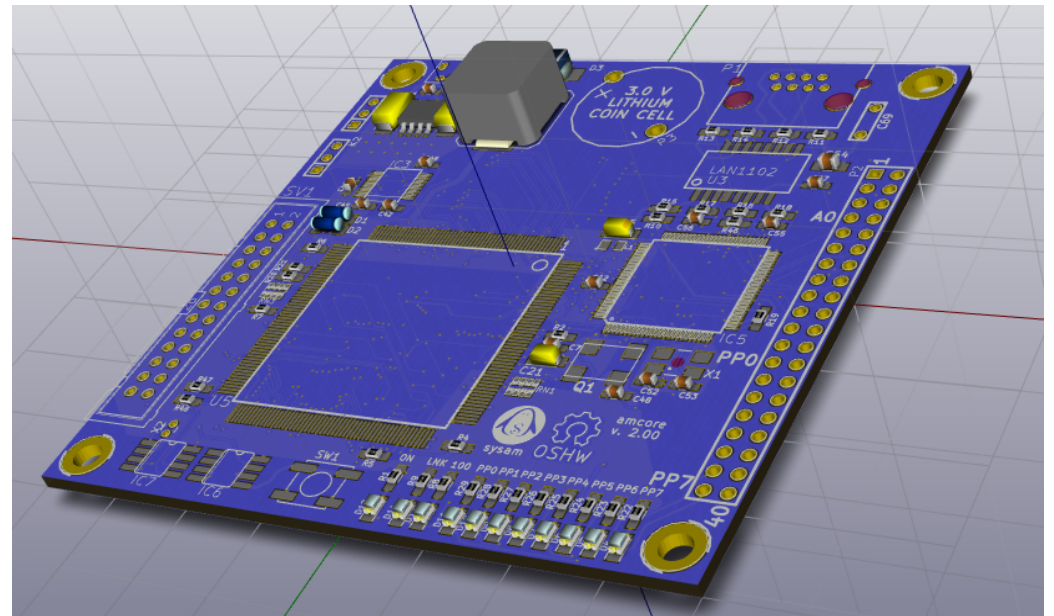
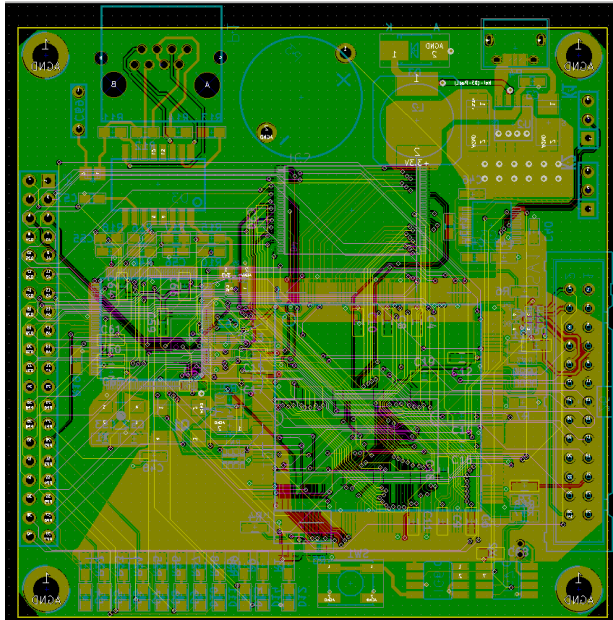


amcore: no terminations, acceptable rings

- as a pragmatic way, use a potentiometer in series in place of resistors, find the good termination value.
- there is much more hardware design theory to improve your knowledge, as you prefer

## PCB DESIGN : AUTO-ROUTING

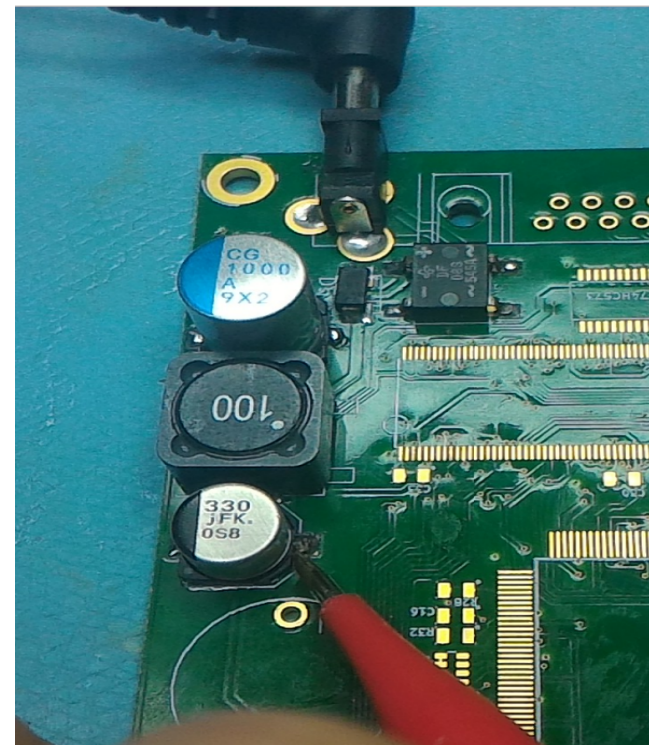
Once the more critical parts are routed as preferred, launching auto-routing over the night can help to find the board quite ready the morning after. Of course some unresolved connection will probably still be there and need to be done by the human ability.



# 4 - BUILDING

## POWER SUPPLY

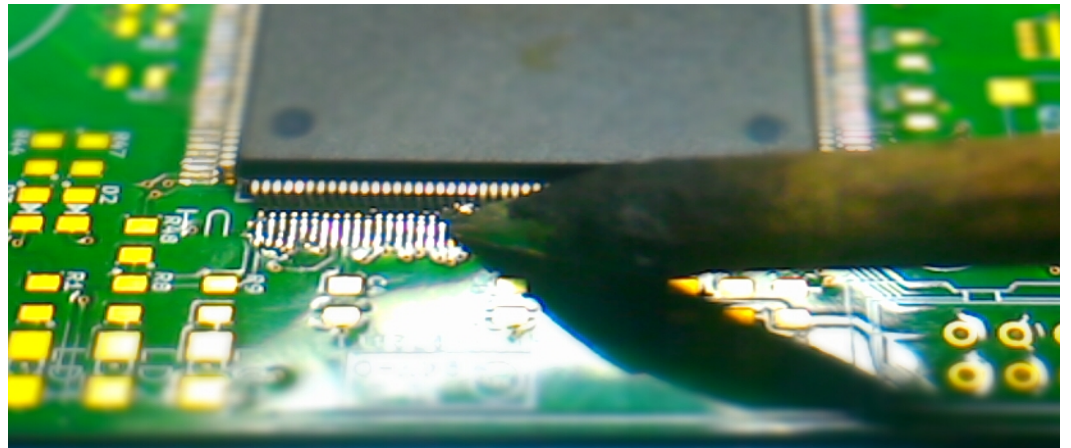
- You can check CPU voltages are correct (tester), without any load, so in case of further shorts soldering components, with voltage that sits down, you know the cause is not the power supply itself.
- A check by scope can help to see if the DC is clean, in case not, check about filtering capacitor ESR and check if some bypass capacitor is needed to remove any unwanted noises.
- On other issues, check the P.S. datasheet and the example schematics in it.



# 4 - BUILDING

## CPU

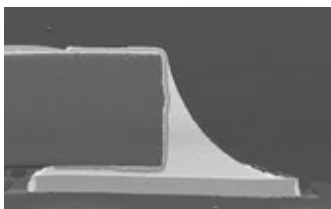
- put it carefully on the PCB, once well centered fix a corner with a small solder joint,
- check all sides to be well aligned, so fix the other corner and side by side solder all the pins.
- drag tin on the right and outside. Use a lot of solder paste, keep shiny solder tip, remove excess of tin in the tip. Solder paste maintains shiny solderings. It will help tin to come away from shorts.
- if solderings are not brilliant, test different solder pastes.



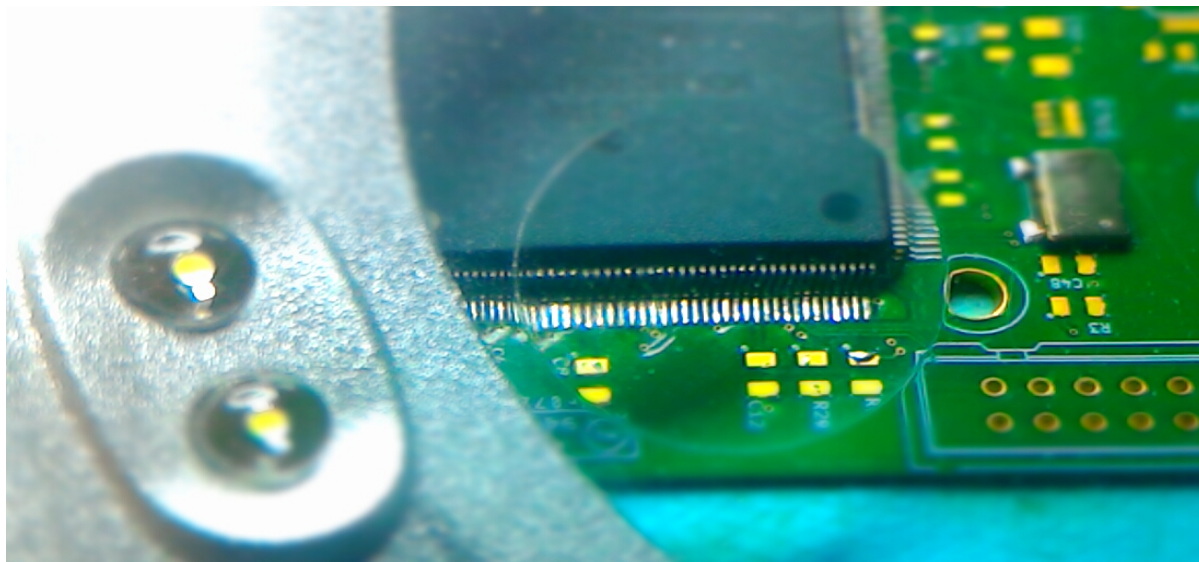
# 4 - BUILDING

## CPU and components around

- verify all the solderings by lens, for shorts or unconnected pin, fix where needed,
  - solder bootstrap resistors, clock oscillator and any other components to have the CPU running,
  - solder debug interface connector,
  - power up again, and verify 3.3V are still there, otherwise check for shorts.
- 
- check soldering shape



mm		mils
1005	-	0402
1608	-	0603
2012	-	0805



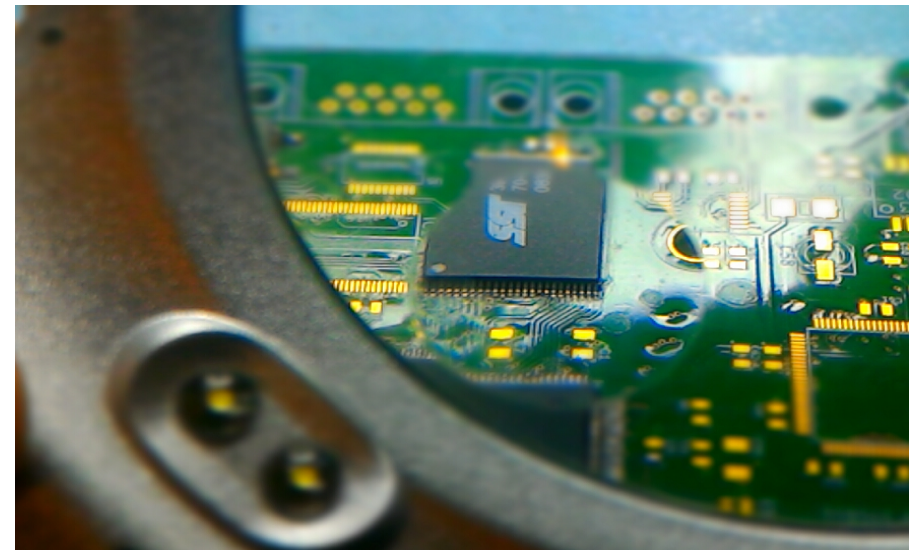
# 4 - BUILDING

## FLASH

- position it, do a temporary fix with tin in a corner, fix it on the other side,
- solder it with same technique used for the CPU (drag right and out), lens check, toothbrush + alcohol cleaning, power up.



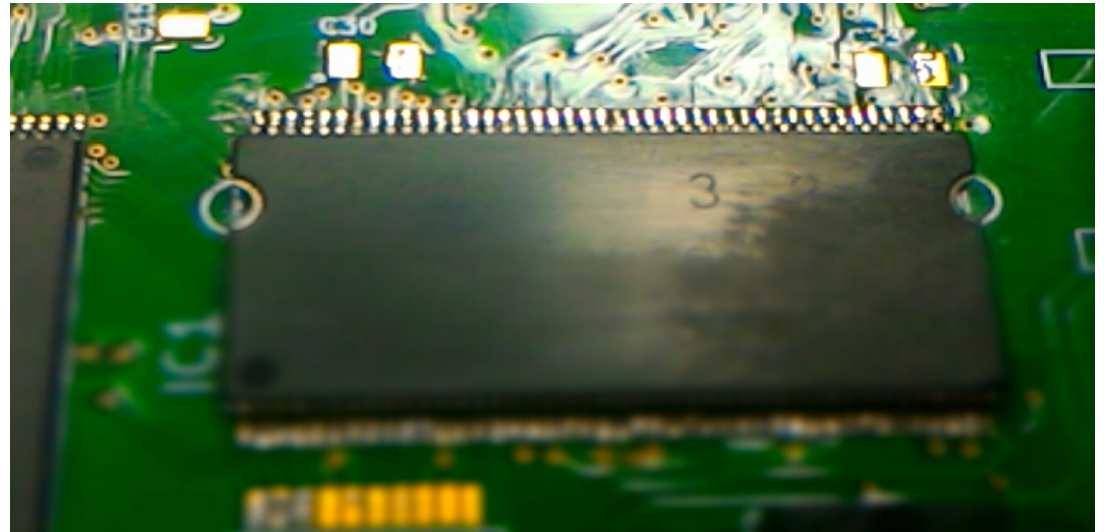
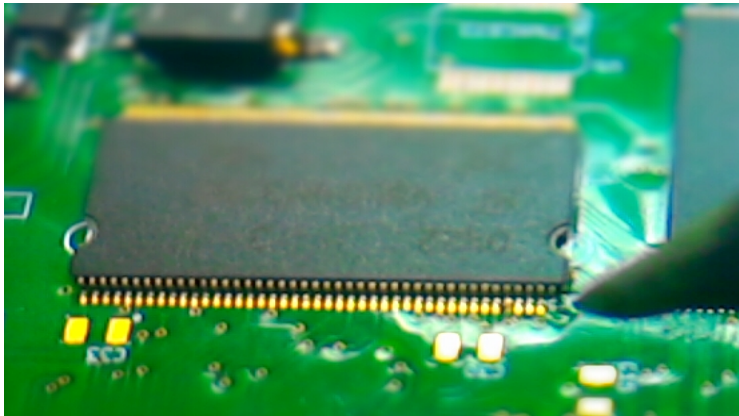
Tip: when there is too much tin, you can heat it and give some rap / shots on the table (no need to hit the board, just hitting the wrist is also good).



# 4 - BUILDING

## SDRAM

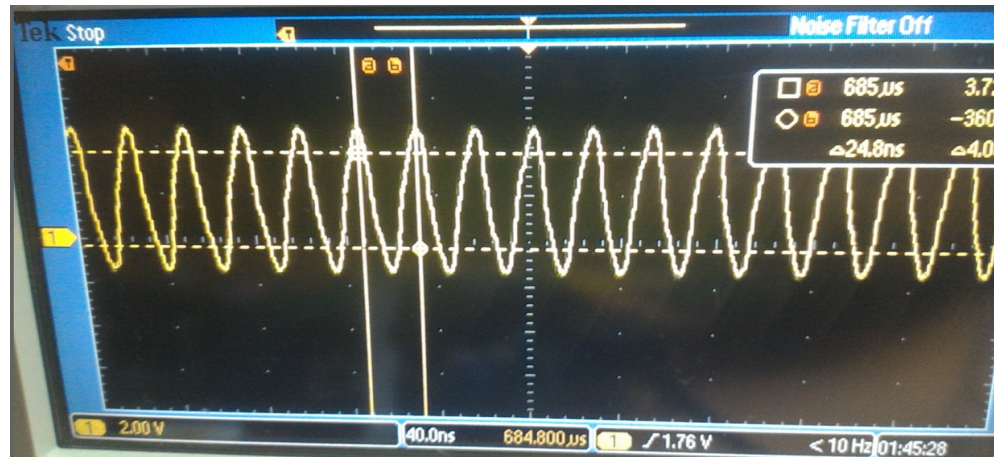
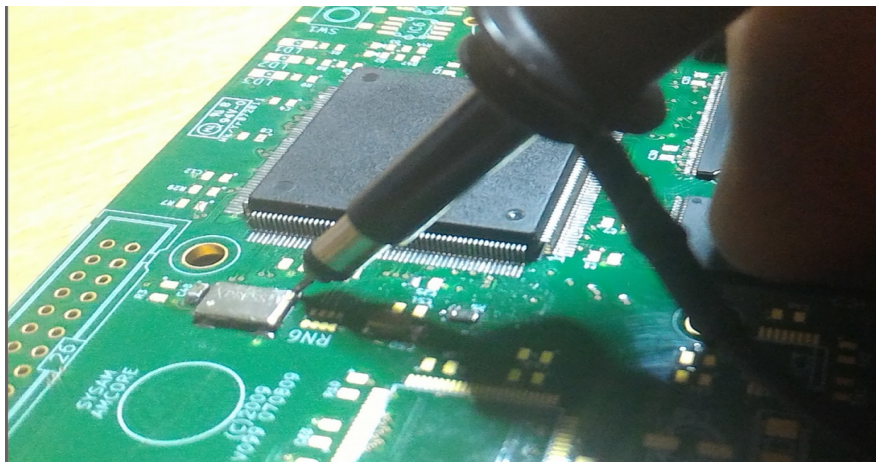
Still same procedure can be used, position – fix – solder, (drag tin right and out), perform lens check, cleaning with toothbrush and alcohol, try to power up and check 3.3V are stable.



# 4 - BUILDING

## FIRST PRELIMINARY CHECKS

- Pin to pin, ohm continuity,
- Board can be powered up. Re verify power is clean and stable (tester / scope on 3V3). Oscillator must be running, a scope check can verify the frequency and the sanity of the waveform shape.



Next step is to connect the debugger / programmer (POD) and try to communicate with the CPU, trying inspecting CPU internal SRAM (static RAM) and / or internal registers.

## PARALLEL NOR FLASH - CHECK AND PROGRAMMING

- connect debugger, Internal CPU SRAM should be now accessible,
- a small binary tool can be loaded into CPU internal SRAM and executed from there trough some simple debuggers commands / scripts,
- whatever tool to program target and debug on target step by step is ok (openocd is a common open solution).
- for the purpose, for the sample “amcore” board only open tools has been used, (gdb + BDM tools have been used for this Coldfire board),
- a cheap hardware debugger / programmer that must be supported by the above tools

# 5 - DEBUG

Purpose of the initial binary tool is to verify the FLASH is well soldered. Tool must fit into internal SRAM of the CPU (4 to 512KB), and must contain some algorithms to erase / program the flash.

```
void flash_unlock_seq()
{
    *(baseaddress + 0x555) = __BE(0xAA);
    *(baseaddress + 0x2AA) = __BE(0x55);
}
```

```
void fl_erase_block (int block)
{
    unsigned long addr = 0x10000 * block / 2;
    // disable interrupts
    asm volatile ("move #0x2700, %sr");

    // erase block (64K)
    flash_unlock_seq();
    *(baseaddress + 0x555) = __BE(0x80);
    flash_unlock_seq();
    *(baseaddress + addr) = __BE(0x30);

    // re-enable interrupts
    asm volatile ("move #0x2000, %sr");

    flash_erase_status_poll(baseaddress + addr);

    delay(0x10000);
}
```

```
/*
 * AMD STANDARD WRITE
 */
void fl_write_word (volatile unsigned short *dest, unsigned short value)
{
    unsigned char ready;

    // disable interrupts
    asm volatile ("move #0x2700, %sr");

    // UNLOCK COMMAND (Software Protection)
    flash_unlock_seq();
    // AMD WRITE COMMAND
    *(baseaddress + 0x555) = __BE(0xA0);

    /* WRITE DATA */
    *dest = value;

    // re-enable interrupts
    asm volatile ("move #0x2000, %sr");

    // Wait for verification...
    while (1) {
        ready = *dest == value;
        /* Good burn! */
        if (ready) break;

        delay(10);
    }
}
```

Also, a minimal UART driver has been prepared for the “amcore” board. It has been used for both debug and firmware upload, to update the flash memory content.

## SDRAM INIT

- before being able to write the Flash, SDRAM should be initialized
- need to program the FLASH at least with a bootloader, so SDRAM correctly initialized is needed to keep an UART-uploaded data into SDRAM before writing it to flash.

```
void meminit(void)
{
    volatile unsigned long *ram = (unsigned long *)0x00000000;

    delay(0x40000);

    /**
     * DCR
     * 0x26 mean Refresh period at 64msec (x4096 rows)
     * 0x8000,0x8100,0x8200 are all valid, faster is 0x8000 (7G, tRC is >=70ns)
     */
    *((volatile unsigned short *) (MCFSIM_DCR)) = 0x8226;

    /**
     * DACR0
     * page mode on burst only (to be verified) 0x00
     * CMD on A20 0x0300
     */
    *((volatile unsigned long *) (MCFSIM_DACR0)) = 0x00003304;
    *((volatile unsigned long *) (MCFSIM_DMR0)) = 0x00ff0001;
    // issue a PRECHARGE ALL
    *((volatile unsigned long *) (MCFSIM_DACR0)) = 0x0000330c;
    *((volatile unsigned long *) (0x00000004)) = 0xbeaddeed;

    *((volatile unsigned long *) (MCFSIM_DACR0)) = 0x0000b304;

    delay(0x68a);

    *((volatile unsigned long *) (0x00000004)) = 0xbeaddeed;
    *((volatile unsigned long *) (MCFSIM_DACR0)) = 0x0000b304;

    delay(0x40000);

    *((volatile unsigned long *) (MCFSIM_DACR0)) = 0x0000b344;
    *((volatile unsigned long *) (0x00000c00)) = 0xbeaddeed;

    // Memory block 1 not in use (DMR1[V] = 0)
    *((volatile unsigned long *) (MCFSIM_DMR1)) = 0x00040000;
    *((volatile unsigned long *) (MCFSIM_DACR1)) = 0x00000000;
}
```

## SDRAM CHECK

- so SDRAM need to be tested with some basic test,
- 1) unfortunate situation, SDRAM is not working, is it an HW or SW problem ?
  - 2) be well sure the initialization sequence is correct  
(see CPU datasheet, application notes and sources around)
  - 3) some test algorithm can be prepared. Lens check, fix and alcohol cleaning should be performed where needed

Try eventually lower bus clock speeds

Stil issues ? Loop to point 1 until this basic test passes.

```
int memtest()
{
    unsigned int i, count=0;
    unsigned long r, *t=0;
    unsigned long pattern;

    write ("\\033[2J\\033[Hmemtest\\r\\n\\r\\n");

    /* get random start */
    pattern = *((volatile unsigned short *) (MCF_MBAR + MCFSIM_TCN0));

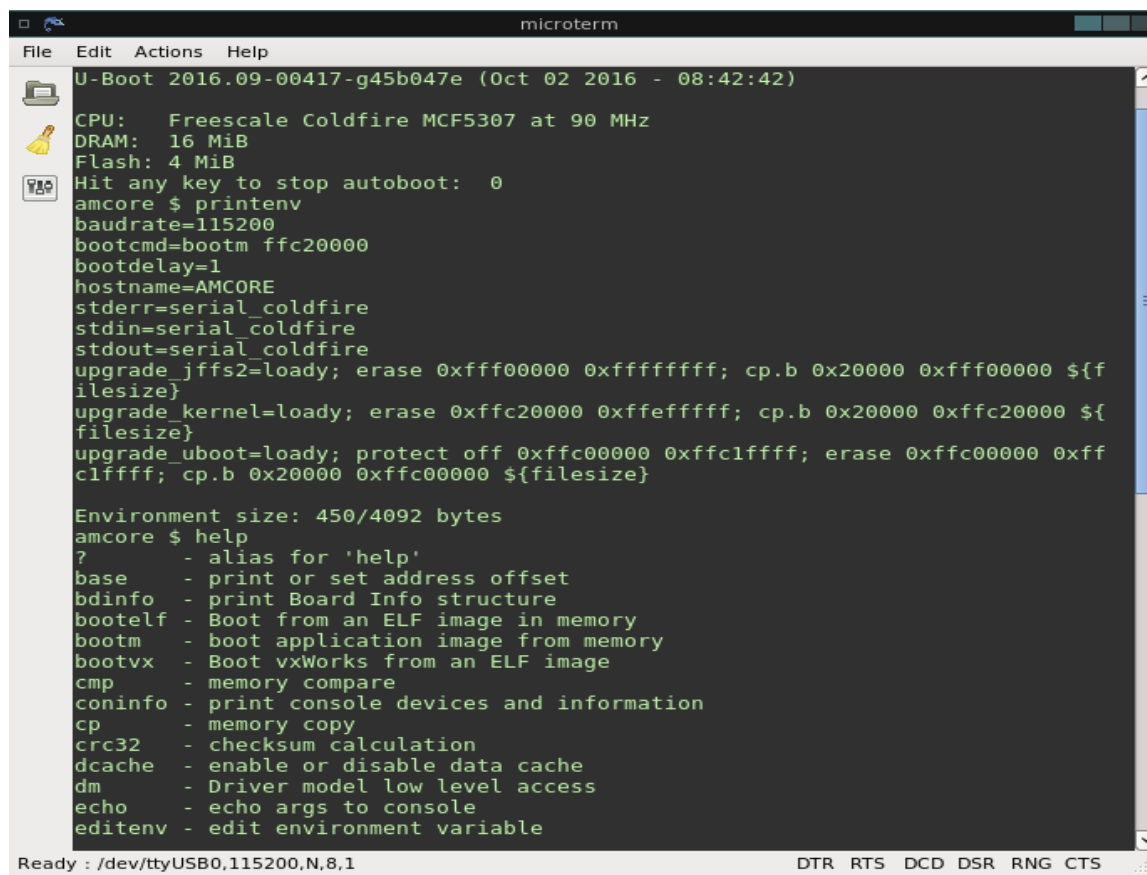
    /**
     * test all over the 16M
     */
    for (i=0; i<SDRAM_SIZE_DW; ++i)
    {
        *t=pattern;
        r=*t++;
        if (r!=pattern++)
        {
            if (count == MAX_ERRORS) {
                write ("Too many errors\\r\\n\\r\\n");
                break;
            }
            err_table[count++][0] = *(pattern-1);
            err_table[count++][1] = r;
        }
        if (i && !(i%SDRAM_1M_BOUNDARY)) write ("\\r\\n");
    }

    if (!count) {
        write ("(k)\\r\\n");
    } else {
        display_errors();
    }
    write ("\\r\\npress...");
    getchar(0);
}
#endif
```

## BOOTLOADER – U-BOOT

In this 4MB NOR parallel flash,  
2 erase sectors of 64KB has been  
reserved for the U-Boot bootloader  
and his environment variables.

Now that all is up and running  
it's just all about software choices.



```

microterm
File Edit Actions Help
U-Boot 2016.09-00417-g45b047e (Oct 02 2016 - 08:42:42)

CPU:   Freescale Coldfire MCF5307 at 90 MHz
DRAM:  16 MiB
Flash: 4 MiB
Hit any key to stop autoboot:  0
amcore $ printenv
baudrate=115200
bootcmd=bootm ffc20000
bootdelay=1
hostname=AMCORE
stderr=serial_coldfire
stdin=serial_coldfire
stdout=serial_coldfire
upgrade_jffs2=loady; erase 0xffff00000 0xffffffff; cp.b 0x20000 0xffff00000 ${f
ilesize}
upgrade_kernel=loady; erase 0xffc20000 0xffefffff; cp.b 0x20000 0xffc20000 ${
filesize}
upgrade_uboot=loady; protect off 0xffc00000 0xffc1ffff; erase 0xffc00000 0xff
c1ffff; cp.b 0x20000 0xffc00000 ${filesize}

Environment size: 450/4092 bytes
amcore $ help
? - alias for 'help'
base - print or set address offset
bdinfo - print Board Info structure
bootelf - Boot from an ELF image in memory
bootm - boot application image from memory
bootvx - Boot vxWorks from an ELF image
cmp - memory compare
coninfo - print console devices and information
cp - memory copy
crc32 - checksum calculation
dcache - enable or disable data cache
dm - Driver model low level access
echo - echo args to console
editenv - edit environment variable

Ready : /dev/ttyUSB0,115200,N,8,1
DTR RTS DCD DSR RNG CTS
  
```

## FINAL STEP, LOADING LINUX

- mtd partitioning, 2 x 64KB sectors has been reserved for u-boot,
- 46 sectors has been reserved for the kernel and a basic ROMFS,
- last 1MB (16 sectors) for a minimal jffs2 data partition

**U-boot, 128KB**

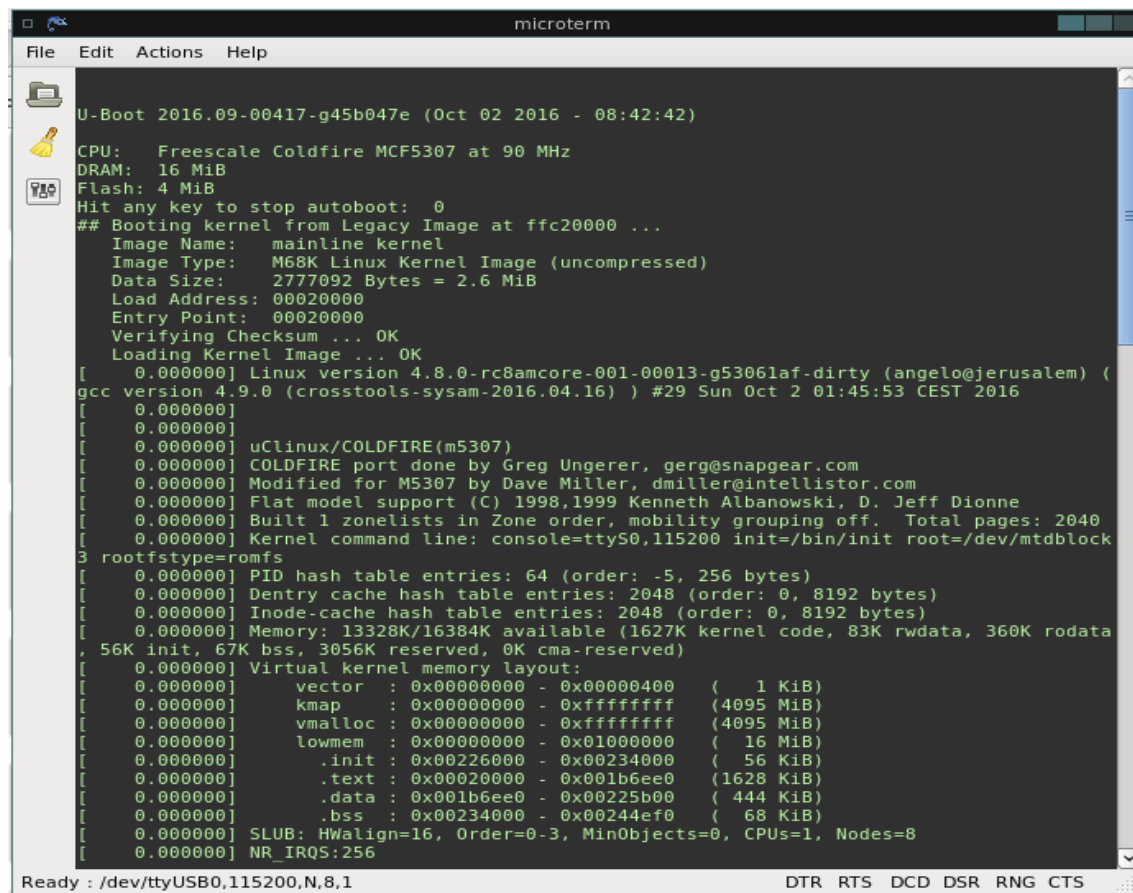
**2,875 MB Kernel + ROMFS**

**1MB JFFS2 RW data partition**

## LOADING LINUX

So, once programmed the kernel partition (can be done easily from a simple u-boot script), and some U-Boot setup, should be possible to see the kernel booting.

```
# cat /proc/cmdline console=ttyS0,115200
init=/bin/init root=/dev/mtdblock3
rootfstype=romfs
```



```

microterm
File Edit Actions Help

U-Boot 2016.09-00417-g45b047e (Oct 02 2016 - 08:42:42)

CPU:   Freescale Coldfire MCF5307 at 90 MHz
DRAM:  16 MiB
Flash:  4 MiB
Hit any key to stop autoboot:  0
## Booting kernel from Legacy Image at ffc20000 ...
   Image Name:   mainline kernel
   Image Type:   M68K Linux Kernel Image (uncompressed)
   Data Size:    2777092 Bytes = 2.6 MiB
   Load Address: 00020000
   Entry Point:  00020000
   Verifying Checksum ... OK
   Loading Kernel Image ... OK
[ 0.000000] Linux version 4.8.0-rc8amcore-001-00013-g53061af-dirty (angelo@jerusalem) (
gcc version 4.9.0 (crosstools-sysam-2016.04.16) ) #29 Sun Oct 2 01:45:53 CEST 2016
[ 0.000000]
[ 0.000000]
[ 0.000000] uClinux/COLDFIRE(m5307)
[ 0.000000] COLDFIRE port done by Greg Ungerer, gerg@snapgear.com
[ 0.000000] Modified for M5307 by Dave Miller, dmiller@intellistor.com
[ 0.000000] Flat model support (C) 1998,1999 Kenneth Albanowski, D. Jeff Dionne
[ 0.000000] Built 1 zonelists in Zone order, mobility grouping off. Total pages: 2040
[ 0.000000] Kernel command line: console=ttyS0,115200 init=/bin/init root=/dev/mtdblock
3 rootfstype=romfs
[ 0.000000] PID hash table entries: 64 (order: -5, 256 bytes)
[ 0.000000] Dentry cache hash table entries: 2048 (order: 0, 8192 bytes)
[ 0.000000] Inode-cache hash table entries: 2048 (order: 0, 8192 bytes)
[ 0.000000] Memory: 13328K/16384K available (1627K kernel code, 83K rwddata, 360K rodata
, 56K init, 67K bss, 3056K reserved, 0K cma-reserved)
[ 0.000000] Virtual kernel memory layout:
[ 0.000000]   vector : 0x00000000 - 0x00000400   ( 1 KiB)
[ 0.000000]   kmap : 0x00000000 - 0xffffffff   (4095 MiB)
[ 0.000000]   vmalloc : 0x00000000 - 0xffffffff   (4095 MiB)
[ 0.000000]   lowmem : 0x00000000 - 0x01000000   ( 16 MiB)
[ 0.000000]   .init : 0x00226000 - 0x00234000   ( 56 KiB)
[ 0.000000]   .text : 0x00020000 - 0x001b6ee0   (1628 KiB)
[ 0.000000]   .data : 0x001b6ee0 - 0x00225b00   ( 444 KiB)
[ 0.000000]   .bss : 0x00234000 - 0x00244ef0   ( 68 KiB)
[ 0.000000] SLUB: HWalign=16, Order=0-3, MinObjects=0, CPUs=1, Nodes=8
[ 0.000000] NR_IRQS:256

Ready : /dev/ttyUSB0,115200,N,8,1
DTR RTS DCD DSR RNG CTS

```

**Thanks to everybody.**

**Special thanks to The Linux Foundation.**

<http://sysam.it/openzone/projects/amcore/amcore.html>

[angelo@sysam.it](mailto:angelo@sysam.it)

# Build Your Own Device

**Angelo Dureghello**  
Sysam Firmware Solutions

Embedded Linux Conference & OpenIoT Summit Europe 2016 - October 11-13th 2016, Berlin, Germany

**Sysam Firmware Solutions**  
[www.sysam.it](http://www.sysam.it)

Linux kernel, device drivers, embedded system design,  
sw development and consulting, hardware prototyping  
and board wake-up support

